

# Pemrograman Berorientasi Objek dengan C#

---

[Lecture Note Pertemuan ke - 14]  
(LINQ)

Ng Poi Wong, 2020, Sesi 7.a : LINQ, Lecture Notes, Pemrograman Berorientasi Objek dengan C# (IF0058), STMIK Mikroskil Medan, Dikirimkan 02 Maret 2020.

Capaian MK : Mahasiswa mampu menerapkan LINQ

**DAFTAR ISI PENJELASAN SLIDE**

Slide 03 s/d 05	[Arsitektur LINQ] .....	2
Slide 06 s/d 09	[Assignment] .....	3
Slide 10	[Let] .....	7
Slide 11 s/d 14	[OrderBy] .....	10
Slide 15 s/d 20	[OrderBy] .....	13
Slide 21 & 22	[Select] .....	16
Slide 23 & 24	[Any & All] .....	19

**PENJELASAN DARI SLIDE ke-03 s/d 05**

- **LINQ (Language-Integrated Query)** merupakan suatu bahasa query yang memudahkan dan menyeragamkan cara manipulasi data.
- Dalam penulisan bahasa query dari SQL, akan berbeda-beda aturan / format penulisan SQL-nya untuk provider Database yang berbeda-beda juga, misalnya aturan / format penulisan SQL untuk provider DBMS SQL Server akan ada perbedaan aturan / format dengan penulisan SQL untuk provider DBMS MySQL, Microsoft Access, DB2, Oracle, atau lainnya.
- Jika menggunakan **LINQ**, maka aturan / format penulisan bahasa querynya akan sama, meskipun provider DBMS-nya berbeda-beda, karena aturan / format penulisan bahasa query dari **LINQ** adalah terintegrasi ke dalam bahasa pemrograman (dalam hal ini adalah C#).
- **LINQ** dapat digunakan untuk mengakses berbagai Data Source, seperti Objects, DataSet, SQL, Entities, dan XML.
- Pada akses **LINQ** ke Data Source berupa DataSet, SQL, dan Entities, dapat dikategorikan sebagai LINQ to ADO.Net, yakni memanfaatkan LINQ untuk mengakses database ADO.NET.
- Pada Sesi ini, akan terfokus pada **LINQ to Objects** untuk mempelajari format penulisan bahasa Query dari LINQ, serta sekilas tentang **LINQ to ADO.NET**.
- **LINQ to Objects** akan terfokus pada Data Source berupa Data Collection (Array atau Koleksi).
- Gambaran struktur dasar dari bahasa Query **LINQ** hampir mirip dengan SQL, bahkan ada sejumlah klausa yang sama dengan SQL, tetapi dengan mekanisme pembacaan yang berbeda.
- Untuk mengambil hasil manipulasi data dari **LINQ**, kita harus menggunakan perulangan **foreach** dan tipe data yang umum digunakan adalah bertipe **var**.
- Dari operator dasar **LINQ**, terdapat klausa umum :
  - **from ...** → Untuk mengakses Data Source yang akan dimanipulasikan. Identik dengan FROM dari SQL.
  - **let ...** → Sebagai alias atau suatu variabel lokal sementara yang hanya berlaku di dalam LINQ itu sendiri. Identik dengan AS dari SQL.
  - **where ...** → Untuk kriteria. Identik dengan WHERE dari SQL.
  - **orderby ...** → Untuk pengurutan. Identik dengan ORDER BY dari SQL.
  - **select ...** → Untuk pengembalian data kepada user (output). Identik dengan SELECT dari SQL.
  - **group ... by ...** → Untuk pengelompokkan. Identik dengan GROUP BY dari SQL.
- Pada bahasa query **LINQ**, terdapat 2 bentuk penulisan **LINQ**, yakni :
  1. Query Syntax
  2. Method Syntax
- **Query Syntax** merupakan bentuk penulisan **LINQ** yang agak panjang, tetapi lebih mudah dipahami. Polanya mirip dengan penulisan SQL pada DBMS, dimana perintahnya dapat dituliskan menjadi banyak baris, dengan masing-masing baris memiliki perannya tersendiri.
- **Method Syntax** merupakan bentuk penulisan **LINQ** yang lebih singkat, tetapi berpotensi membingungkan. Polanya seperti memanggil Method dari suatu Objek, dan dapat dilanjutkan dengan sub-method dan seterusnya.
- Dari semua klausa yang ada pada **LINQ**, terdapat sejumlah klausa yang dapat digunakan secara **Query Syntax** maupun **Method Syntax**, tetapi ada beberapa klausa yang HANYA tersedia untuk **Query Syntax** saja atau **Method Syntax** saja.
- Pada prakteknya, manipulasi data dengan LINQ, dapat kita kombinasikan antara **Query Syntax** dan **Method Syntax**.

**PENJELASAN DARI SLIDE ke-06 s/d 09**

**[Slide 06 s/d 08]**

- **Assignment** merupakan operator dasar untuk penugasan.
- Wajib menggunakan klausa **from** dan **select**.
- Klausa **from** untuk membaca / mengambil Data Source yang akan dimanipulasikan, dengan format klausa :  
`from x in y`  
 dimana **x** adalah variabel alias yang mewakili setiap data / record dari sumber data **y** (Data Source).
- Klausa **select** untuk assignment hasil manipulasi / query ke suatu media penyimpanan (dapat berupa variabel), dimana tipe data untuk menampung hasil query (return value) adalah bertipe **var**.
- Tipe data **var** untuk menampung hasil query LINQ merupakan tipe data universal, dimana dapat menampung tipe data apapun.
- Apabila ingin menyatakan tipe data tertentu untuk menyimpan hasil query LINQ, maka dapat menggunakan interface **IEnumerable<T>**, dimana **T** untuk menyatakan tipe data dari data yang di-return dari query.
- Tipe data **var** identik dengan tipe data koleksi **ArrayList**, sedangkan interface **IEnumerable<T>** identik dengan tipe data koleksi **List** (Lihat **Sesi 6 - Array & Koleksi** dari MK Pengantar Pemrograman C# dari semester sebelumnya).

**[Slide 09]**

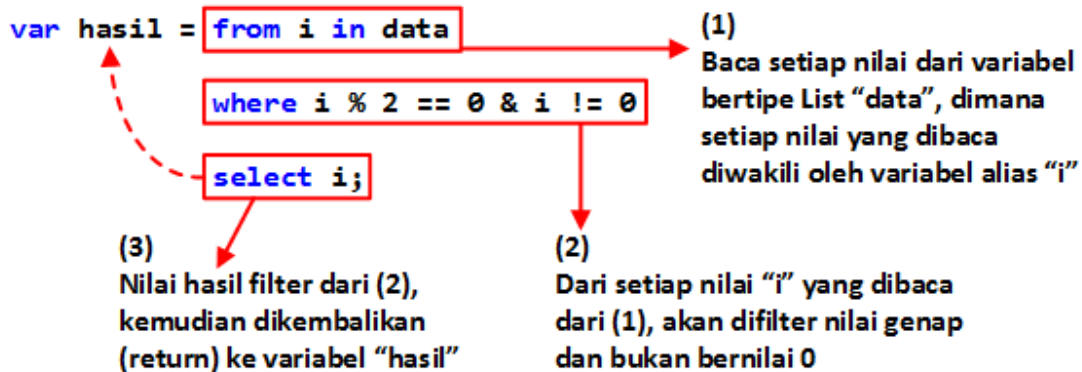
- Klausa **where** merupakan operator dasar untuk kriteria.
- Klausa **where** dapat dituliskan secara Query Syntax dan Method Syntax.

**[Contoh Sederhana]**

- Contoh sederhana dari penggunaan klausa **from**, **select**, dan **where** adalah menggunakan Console Application.
- Untuk **Contoh Sederhana ke-1**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            where i % 2 == 0 & i != 0
            select i;
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Query Syntax, dengan penjelasan berikut :

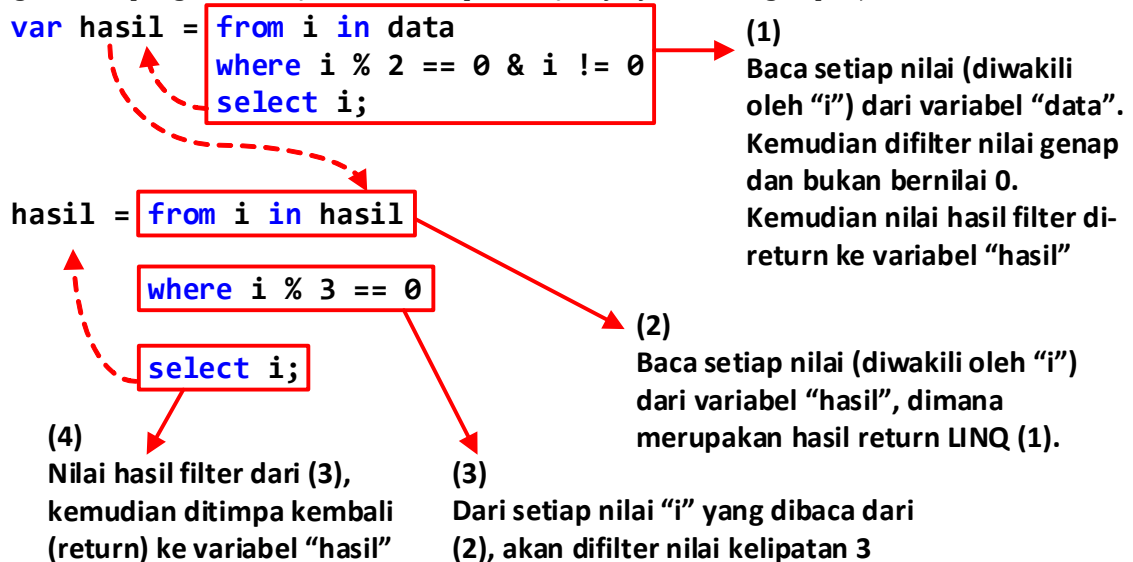


**PENJELASAN DARI SLIDE ke-06 s/d 09**

- Untuk **Contoh Sederhana ke-2**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            where i % 2 == 0 & i != 0
            select i;
hasil = from i in hasil
        where i % 3 == 0
        select i;
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Query Syntax, dengan penjelasan berikut :



- Untuk **Contoh Sederhana ke-3**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
IEnumerable<int> hasil = from i in data
                        where i % 2 == 0 & i != 0
                        select i;

foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas adalah sama dengan **Contoh Sederhana ke-1** di atas, dengan perbedaan pada deklarasi tipe data dari variabel "hasil" untuk menampung hasil LINQ, dimana variabel "hasil" bertipe `IEnumerable<int>` yang menyatakan nilai kembalian dari LINQ adalah bertipe `int`.

**PENJELASAN DARI SLIDE ke-06 s/d 09**

- Untuk **Contoh Sederhana ke-4**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = data.Where(i => i % 2 != 0);
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

```
var hasil = data.Where(i => i % 2 != 0);
```

Setiap nilai dari variabel bertipe **List "data"**, kemudian difilter dengan method **Where**.

Penulisan method **"Where(i => i % 2 != 0)"** dibaca : Lakukan filter untuk setiap nilai yang diwakili **"i"**, dimana nilai **"i"** adalah bilangan ganjil.

- Untuk **Contoh Sederhana ke-5**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = data.Where(i => i % 2 == 0 & i != 0).Where(n => n % 3 == 0);
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

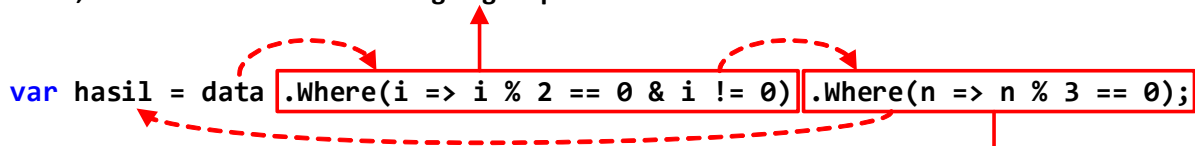
Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

(1)

Setiap nilai dari variabel **"data"**, difilter dengan method **Where**.

Method **"Where(i => i % 2 == 0 & i != 0)"** dibaca :

Lakukan filter untuk setiap nilai dari variabel **"data"** yang diwakili **"i"**, dimana nilai **"i"** adalah bilangan genap dan bukan bernilai 0.



```
var hasil = data.Where(i => i % 2 == 0 & i != 0).Where(n => n % 3 == 0);
```

(2)

Hasil filter dengan method **Where** dari (1), kemudian difilter lagi dengan method **Where**.

Method **"Where(n => n % 3 == 0)"** dibaca : Lakukan filter untuk setiap nilai dari hasil filter (1) yang diwakili **"n"**, dimana nilai **"n"** adalah bilangan kelipatan 3.

Hasil filter yang ke-2 ini, kemudian dikembalikan (return) ke variabel **"hasil"**.

- Untuk **Contoh Sederhana ke-6**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
IEnumerable<int> hasil = data.Where(i => i % 2 != 0);
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas adalah sama dengan **Contoh Sederhana ke-4** di atas, dengan perbedaan pada deklarasi tipe data dari variabel **"hasil"** untuk menampung hasil LINQ, dimana variabel **"hasil"** bertipe **IEnumerable<int>** yang menyatakan nilai kembalian dari LINQ adalah bertipe **int**.

**PENJELASAN DARI SLIDE ke-06 s/d 09**

- Untuk **Contoh Sederhana ke-7**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };  
foreach (int i in data.Where(i => i % 2 != 0))  
    Console.WriteLine("{0} ", i);
```

Penulisan LINQ secara Method Syntax, dapat dikombinasikan dengan statement kode program lainnya. Pada contoh ini, penulisan LINQ secara Method Syntax dikombinasikan dengan statement perulangan foreach, dimana setiap nilai dari variabel **"data"** yang telah didifilter untuk nilai ganjil, langsung dikembalikan ke perulangan foreach.

## PENJELASAN DARI SLIDE ke-10

- Klausula **let** merupakan operator dasar untuk mendefinisikan suatu variabel lokal dan menugaskan suatu ekspresi ke dalamnya. Variabel lokal ini hanya dapat digunakan di dalam ruang lingkup pernyataan LINQ yang mendefinisikan variabel lokal tersebut.
- Klausula **let** hanya dapat digunakan pada Query Syntax.

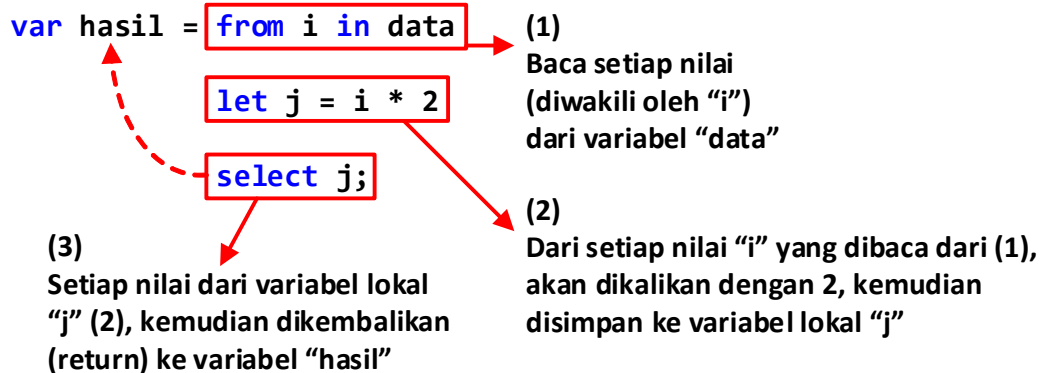
### [Contoh Sederhana]

- Contoh sederhana dari penggunaan klausula **let** adalah menggunakan Console Application.

- Untuk **Contoh Sederhana ke-1**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            let j = i * 2
            select j;
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :



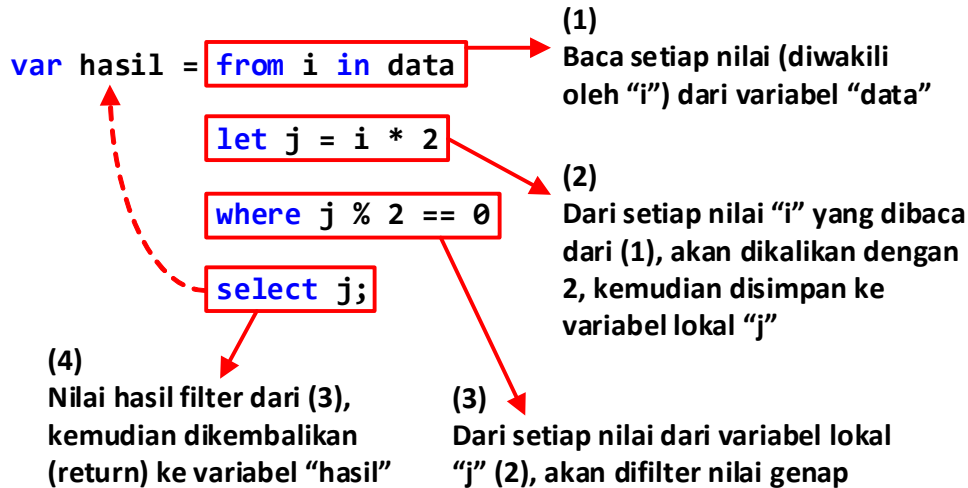
- Untuk **Contoh Sederhana ke-2**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            let j = i * 2
            where j % 2 == 0
            select j;
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```



## PENJELASAN DARI SLIDE ke-10

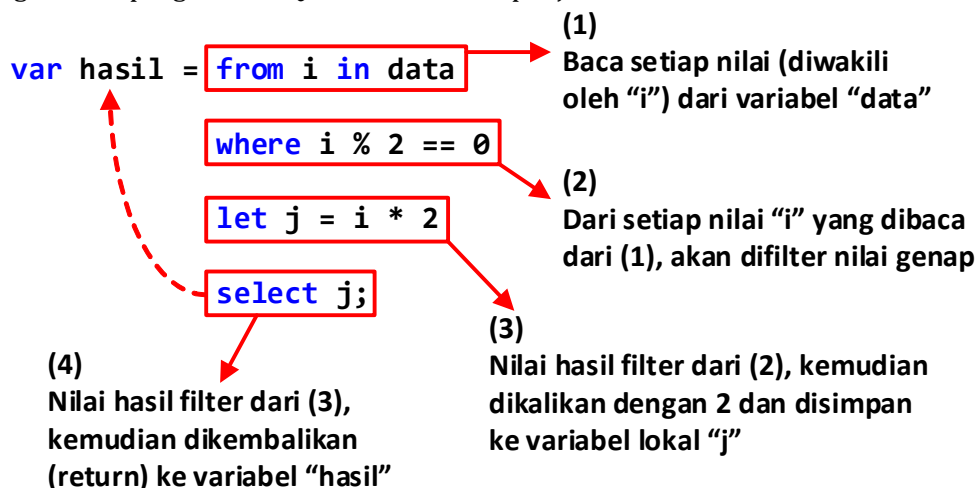
Potongan kode program LINQ di atas memiliki penjelasan berikut :



➤ Untuk Contoh Sederhana ke-3, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            where i % 2 == 0
            let j = i * 2
            select j;
foreach (int i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

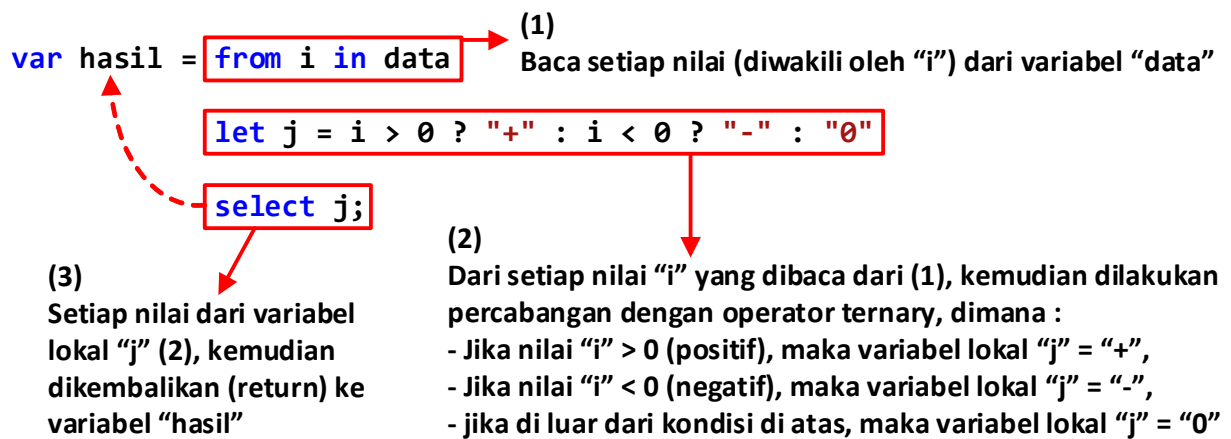


## PENJELASAN DARI SLIDE ke-10

- Untuk **Contoh Sederhana ke-4**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil = from i in data
            let j = i > 0 ? "+" : i < 0 ? "-" : "0"
            select j;
foreach (string i in hasil)
    Console.WriteLine("{0} ", i);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :



**PENJELASAN DARI SLIDE ke-11 s/d 14**

- Klausula **orderby** merupakan operator dasar untuk pengurutan data.
- Klausula **orderby** dapat dituliskan secara Query Syntax dan Method Syntax.
- Pengurutan dengan klausula **orderby** dapat dilakukan secara Ascending maupun Descending, dimana secara default adalah diurutkan secara Ascending.
- Untuk pengurutan secara Descending secara Query Syntax, wajib ditambahkan keyword **descending**.
- Untuk pengurutan secara Descending secara Method Syntax, digunakan method **OrderByDescending**.

**[Contoh Sederhana]**

- Contoh sederhana dari penggunaan klausula **orderby** adalah menggunakan Console Application.
- Untuk **Contoh Sederhana ke-1**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil1 = from x in data
              orderby x
              select x;
var hasil2 = from x in data
              orderby x descending
              select x;
foreach (int i in hasil1)
    Console.Write("{0} ", i);
Console.WriteLine();
foreach (int i in hasil2)
    Console.Write("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Query Syntax, dengan penjelasan berikut :

1. Untuk potongan LINQ pertama :

```
var hasil1 = from x in data
              orderby x
              select x;
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**data**",

Kemudian diurutkan secara Ascending,

Kemudian hasil pengurutan tersebut dikembalikan (return) ke variabel "**hasil1**".

2. Untuk potongan LINQ ke-2 :

```
var hasil2 = from x in data
              orderby x descending
              select x;
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**data**",

Kemudian diurutkan secara Descending,

Kemudian hasil pengurutan tersebut dikembalikan (return) ke variabel "**hasil2**".

## PENJELASAN DARI SLIDE ke-11 s/d 14

- Untuk **Contoh Sederhana ke-2**, diberikan kode program berikut :

```
List<string> nama = new List<string>()
{ "Sally", "Alex", "Meri", "Yohan", "Donita" };
var hasil1 = from x in nama
              where x.ToUpper().Contains('A')
              orderby x
              select x;
var hasil2 = from x in nama
              orderby x.Substring(2, 1) descending
              select x;
foreach (string i in hasil1)
    Console.Write("{0} ", i);
Console.WriteLine();
foreach (string i in hasil2)
    Console.Write("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Query Syntax, dengan penjelasan berikut :

1. Untuk potongan LINQ pertama :

```
var hasil1 = from x in nama
              where x.ToUpper().Contains('A')
              orderby x
              select x;
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**nama**",  
Kemudian difilter dengan proses : setiap nilai dari "x" diubah menjadi kapital, kemudian diambil nilai dari "x" yang mengandung karakter "A",  
Kemudian hasil filter tersebut diurutkan secara Ascending,  
Kemudian hasil pengurutan tersebut dikembalikan (return) ke variabel "**hasil1**".

2. Untuk potongan LINQ ke-2 :

```
var hasil2 = from x in nama
              orderby x.Substring(2, 1) descending
              select x;
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**nama**",  
Kemudian diurutkan berdasarkan karakter ke-3 dari nilai "x" secara Descending,  
Kemudian hasil pengurutan tersebut dikembalikan (return) ke variabel "**hasil2**".

- Untuk **Contoh Sederhana ke-3**, diberikan kode program berikut :

```
List<int> data = new List<int>() { 21, 12, -5, 18, -33, 0, 22, 19, 4, 24 };
var hasil1 = data.OrderBy(x => x);
var hasil2 = data.OrderByDescending(x => x);
foreach (int i in hasil1)
    Console.Write("{0} ", i);
Console.WriteLine();
foreach (int i in hasil2)
    Console.Write("{0} ", i);
```

## PENJELASAN DARI SLIDE ke-11 s/d 14

Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

1. Untuk potongan LINQ pertama :

```
var hasil1 = data.OrderBy(x => x);
```

Fungsi dari potongan LINQ ini adalah sama dengan **potongan LINQ pertama dari Contoh Sederhana ke-1**, dengan perbedaan ada pada bentuk penulisan LINQ.

Penulisan method "**OrderBy(x => x)**" dibaca : Lakukan pengurutan secara Ascending untuk setiap nilai yang diwakili "x".

2. Untuk potongan LINQ ke-2 :

```
var hasil2 = data.OrderByDescending(x => x);
```

Fungsi dari potongan LINQ ini adalah sama dengan **potongan LINQ ke-2 dari Contoh Sederhana ke-1**, dengan perbedaan ada pada bentuk penulisan LINQ.

Penulisan method "**OrderByDescending(x => x)**" dibaca : Lakukan pengurutan secara Descending untuk setiap nilai yang diwakili "x".

- Untuk **Contoh Sederhana ke-4**, diberikan kode program berikut :

```
List<string> nama = new List<string>()
{ "Sally", "Alex", "Meri", "Yohan", "Donita" };
var hasil1 = nama.Where(x => x.ToUpper().Contains('A')).OrderBy(x => x);
var hasil2 = nama.OrderByDescending(x => x.Substring(2, 1));
foreach (string i in hasil1)
    Console.Write("{0} ", i);
Console.WriteLine();
foreach (string i in hasil2)
    Console.Write("{0} ", i);
```

Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

1. Untuk potongan LINQ pertama :

```
var hasil1 = nama.Where(x => x.ToUpper().Contains('A')).OrderBy(x => x);
```

Fungsi dari potongan LINQ ini adalah sama dengan **potongan LINQ pertama dari Contoh Sederhana ke-2**, dengan perbedaan ada pada bentuk penulisan LINQ.

Penulisan method "**Where(x => x.ToUpper().Contains('A'))**" dibaca : Lakukan filter untuk setiap nilai yang diwakili "x", dimana nilai "x" diubah menjadi kapital, kemudian diambil nilai dari "x" yang mengandung karakter "A".

Penulisan method "**OrderBy(x => x)**" dibaca : Lakukan pengurutan secara Ascending untuk hasil filter dari nilai yang diwakili "x".

2. Untuk potongan LINQ ke-2 :

```
var hasil2 = nama.OrderByDescending(x => x.Substring(2, 1));
```

Fungsi dari potongan LINQ ini adalah sama dengan **potongan LINQ ke-2 dari Contoh Sederhana ke-2**, dengan perbedaan ada pada bentuk penulisan LINQ.

Penulisan method "**OrderByDescending(x => x.Substring(2, 1))**" dibaca : Lakukan pengurutan berdasarkan karakter ke-3 dari setiap nilai yang diwakili "x" secara Descending.

**PENJELASAN DARI SLIDE ke-15 s/d 20**

- Pengurutan pada LINQ dapat dilakukan pengurutan secara bertingkat, dimana jika ditemukan 2 atau lebih nilai yang sama saat pengurutan, dapat dilanjutkan ke pengurutan tingkat berikutnya.
- Untuk pengurutan bertingkat pada LINQ, dapat diatur jenis pengurutan yang berbeda-beda untuk setiap tingkatan, misalnya tingkat pertama secara Ascending, tingkat ke-2 secara Descending, dan seterusnya.
- Pada penulisan secara Query Syntax, pernyataan pengurutan bertingkat digunakan tanda baca “,” (koma).
- Pada penulisan secara Method Syntax, pernyataan pengurutan bertingkat digunakan method **ThenBy** untuk pengurutan secara Ascending, dan method **ThenByDescending** untuk pengurutan secara Descending.

**[Contoh Sederhana]**

- Contoh sederhana dari penggunaan klausa **orderby** untuk pengurutan bertingkat adalah menggunakan Console Application.
- Deklarasikan struktur berikut pada Console Application :

```
struct Produk {  
    public int Kode;  
    public string Nama, Jenis;  
}
```

- Untuk **Contoh Sederhana ke-1**, diberikan kode program berikut :

```
List<Produk> barang = new List<Produk>() {  
    new Produk() {Kode = 65, Nama = "Gula", Jenis = "Sembako"},  
    new Produk() {Kode = 12, Nama = "Kemeja", Jenis = "Pakaian"},  
    new Produk() {Kode = 98, Nama = "Beras", Jenis = "Sembako"},  
    new Produk() {Kode = 75, Nama = "Jaket", Jenis = "Pakaian"},  
    new Produk() {Kode = 34, Nama = "Spidol", Jenis = "ATK"},  
    new Produk() {Kode = 67, Nama = "Garam", Jenis = "Sembako"},  
    new Produk() {Kode = 41, Nama = "Kertas", Jenis = "ATK"},  
    new Produk() {Kode = 86, Nama = "Pulpen", Jenis = "ATK"},  
    new Produk() {Kode = 73, Nama = "Jeans", Jenis = "Pakaian"},  
    new Produk() {Kode = 92, Nama = "Pensil", Jenis = "ATK"}  
};  
var hasil = from i in barang  
            orderby i.Jenis descending, i.Nama  
            select i;  
foreach (Produk i in hasil)  
    Console.WriteLine("{0,-3:D}{1,-8}{2}", i.Kode, i.Nama, i.Jenis);
```

**PENJELASAN DARI SLIDE ke-15 s/d 20**

Potongan kode program LINQ di atas merupakan Query Syntax, dengan penjelasan berikut :

```
var hasil = from i in barang
            orderby i.Jenis descending, i.Nama
            select i;
```

Baca setiap nilai (diwakili oleh "i") dari variabel "**barang**", dimana pada setiap nilai "i" akan terdiri dari 3 (tiga) variabel, yakni variabel "**Kode**", "**Nama**", dan "**Jenis**",

Kemudian dilakukan pengurutan bertingkat sebanyak 2 (dua) tingkat, dimana tingkat pertama diurutkan berdasarkan variabel "**Jenis**" secara Descending. Jika terdapat 2 atau lebih nilai variabel "**Jenis**" yang bernilai sama, maka berikutnya dilanjutkan ke tingkat ke-2, dimana akan diurutkan berdasarkan variabel "**Nama**" secara Ascending.

Kemudian hasil pengurutan tersebut dikembalikan (return) ke variabel "**hasil**".

➤ Untuk **Contoh Sederhana ke-2**, diberikan kode program berikut :

```
List<Produk> barang = new List<Produk>() {
    new Produk() {Kode = 65, Nama = "Gula", Jenis = "Sembako"},
    new Produk() {Kode = 12, Nama = "Kemeja", Jenis = "Pakaian"},
    new Produk() {Kode = 98, Nama = "Beras", Jenis = "Sembako"},
    new Produk() {Kode = 75, Nama = "Jaket", Jenis = "Pakaian"},
    new Produk() {Kode = 34, Nama = "Spidol", Jenis = "ATK"},
    new Produk() {Kode = 67, Nama = "Garam", Jenis = "Sembako"},
    new Produk() {Kode = 41, Nama = "Kertas", Jenis = "ATK"},
    new Produk() {Kode = 86, Nama = "Pulpen", Jenis = "ATK"},
    new Produk() {Kode = 73, Nama = "Jeans", Jenis = "Pakaian"},
    new Produk() {Kode = 92, Nama = "Pensil", Jenis = "ATK"}
};
var hasil = barang.OrderBy(i => i.Jenis).ThenByDescending(i => i.Nama);
foreach (Produk i in hasil)
    Console.WriteLine("{0,-3:D}{1,-8}{2}", i.Kode, i.Nama, i.Jenis);
```

Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

```
var hasil = barang.OrderBy(i => i.Jenis).ThenByDescending(i => i.Nama);
```

Dilakukan pengurutan bertingkat sebanyak 2 (dua) tingkat terhadap nilai dari variabel "**barang**", dimana :

- Tingkat pertama dilakukan pengurutan dengan penulisan method "**OrderBy(i => i.Jenis)**" dibaca : Lakukan pengurutan secara Ascending untuk setiap nilai yang diwakili "i" berdasarkan variabel "**Jenis**".
- Jika terdapat 2 atau lebih nilai variabel "**Jenis**" yang bernilai sama, maka berikutnya dilanjutkan ke tingkat ke-2, dimana akan diurutkan dengan penulisan method "**ThenByDescending(i => i.Nama)**" dibaca : Lakukan pengurutan secara Descending untuk setiap nilai yang diwakili "i" berdasarkan variabel "**Nama**".

## PENJELASAN DARI SLIDE ke-15 s/d 20

- Untuk **Contoh Sederhana ke-3**, diberikan kode program berikut :

```
List<Produk> barang = new List<Produk>() {
    new Produk() {Kode = 65, Nama = "Gula", Jenis = "Sembako"},
    new Produk() {Kode = 12, Nama = "Kemeja", Jenis = "Pakaian"},
    new Produk() {Kode = 98, Nama = "Beras", Jenis = "Sembako"},
    new Produk() {Kode = 75, Nama = "Jaket", Jenis = "Pakaian"},
    new Produk() {Kode = 34, Nama = "Spidol", Jenis = "ATK"},
    new Produk() {Kode = 67, Nama = "Garam", Jenis = "Sembako"},
    new Produk() {Kode = 41, Nama = "Kertas", Jenis = "ATK"},
    new Produk() {Kode = 86, Nama = "Pulpen", Jenis = "ATK"},
    new Produk() {Kode = 73, Nama = "Jeans", Jenis = "Pakaian"},
    new Produk() {Kode = 92, Nama = "Pensil", Jenis = "ATK"}
};
var hasil = barang.OrderByDescending(i => i.Jenis).ThenBy(i => i>Nama);
foreach (Produk i in hasil)
    Console.WriteLine("{0,-3:D}{1,-8}{2}", i.Kode, i>Nama, i.Jenis);
```

Potongan kode program LINQ di atas merupakan Method Syntax, dengan penjelasan berikut :

```
var hasil = barang.OrderByDescending(i => i.Jenis).ThenBy(i => i>Nama);
```

Dilakukan pengurutan bertingkat sebanyak 2 (dua) tingkat terhadap nilai dari variabel "**barang**", dimana :

- c. Tingkat pertama dilakukan pengurutan dengan penulisan method "**OrderByDescending(i => i.Jenis)**" dibaca : Lakukan pengurutan secara Descending untuk setiap nilai yang diwakili "**i**" berdasarkan variabel "**Jenis**".
- d. Jika terdapat 2 atau lebih nilai variabel "**Jenis**" yang bernilai sama, maka berikutnya dilanjutkan ke tingkat ke-2, dimana akan diurutkan dengan penulisan method "**ThenBy(i => i>Nama)**" dibaca : Lakukan pengurutan secara Ascending untuk setiap nilai yang diwakili "**i**" berdasarkan variabel "**Nama**".



## PENJELASAN DARI SLIDE ke-21 & 22

- Klausula **select** merupakan operator dasar untuk assignment hasil manipulasi / query ke suatu media penyimpanan (dapat berupa variabel).
- Pada slide sebelumnya, telah dijelaskan penggunaan klausula **select** yang dipadukan dengan klausula **from**, dimana klausula **select** mengembalikan hasil query ke variabel bertipe **var**.
- Dengan memanfaatkan klausula **select**, dapat juga mengembalikan suatu nilai yang disesuaikan dengan keinginan user.
- Nilai yang dikembalikan dari query LINQ yang sesuai dengan keinginan user, dapat berupa 1 (satu) atau banyak nilai, baik nilai yang bersumber dari Data Source, maupun suatu nilai yang baru yang merupakan hasil dari suatu proses perhitungan di dalam LINQ yang sedang berjalan.

### [Contoh Sederhana]

- Contoh sederhana dari penggunaan lanjutan dari klausula **select** adalah menggunakan Console Application.
- Deklarasikan struktur berikut pada Console Application untuk **Contoh Sederhana ke-1** dan **Contoh Sederhana ke-2** :

```
struct Produk {
    public int Kode;
    public string Nama, Jenis;
}
```

- Untuk **Contoh Sederhana ke-1**, diberikan kode program berikut :

```
List<Produk> barang = new List<Produk>() {
    new Produk() {Kode = 65, Nama = "Gula", Jenis = "Sembako"},
    new Produk() {Kode = 12, Nama = "Kemeja", Jenis = "Pakaian"},
    new Produk() {Kode = 98, Nama = "Beras", Jenis = "Sembako"},
    new Produk() {Kode = 75, Nama = "Jaket", Jenis = "Pakaian"},
    new Produk() {Kode = 34, Nama = "Spidol", Jenis = "ATK"},
    new Produk() {Kode = 67, Nama = "Garam", Jenis = "Sembako"},
    new Produk() {Kode = 41, Nama = "Kertas", Jenis = "ATK"},
    new Produk() {Kode = 86, Nama = "Pulpen", Jenis = "ATK"},
    new Produk() {Kode = 73, Nama = "Jeans", Jenis = "Pakaian"},
    new Produk() {Kode = 92, Nama = "Pensil", Jenis = "ATK"}
};
var hasil = from x in barang
            orderby x.Nama
            select x.Nama;
foreach (string i in hasil)
    Console.WriteLine(i);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

```
var hasil = from x in barang
            orderby x.Nama
            select x.Nama;
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**barang**", dimana pada setiap nilai "**x**" akan terdiri dari 3 (tiga) variabel, yakni variabel "**Kode**", "**Nama**", dan "**Jenis**",  
Kemudian diurutkan berdasarkan variabel "**Nama**" dari nilai "**x**" secara Ascending,  
Kemudian hasil pengurutan tersebut hanya dikembalikan (return) variabel "**Nama**" dari nilai "**x**" ke variabel "**hasil**".

## PENJELASAN DARI SLIDE ke-21 & 22

- Untuk **Contoh Sederhana ke-2**, diberikan kode program berikut :

```
List<Produk> barang = new List<Produk>() {
    new Produk() {Kode = 65, Nama = "Gula", Jenis = "Sembako"},
    new Produk() {Kode = 12, Nama = "Kemeja", Jenis = "Pakaian"},
    new Produk() {Kode = 98, Nama = "Beras", Jenis = "Sembako"},
    new Produk() {Kode = 75, Nama = "Jaket", Jenis = "Pakaian"},
    new Produk() {Kode = 34, Nama = "Spidol", Jenis = "ATK"},
    new Produk() {Kode = 67, Nama = "Garam", Jenis = "Sembako"},
    new Produk() {Kode = 41, Nama = "Kertas", Jenis = "ATK"},
    new Produk() {Kode = 86, Nama = "Pulpen", Jenis = "ATK"},
    new Produk() {Kode = 73, Nama = "Jeans", Jenis = "Pakaian"},
    new Produk() {Kode = 92, Nama = "Pensil", Jenis = "ATK"}
};
var hasil = from x in barang
            orderby x.Kode
            select new {
                Kode = x.Kode,
                Nama = x.Nama
            };
foreach (var i in hasil)
    Console.WriteLine("{0,-3:D}{1,-8}", i.Kode, i.Nama);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

```
var hasil = from x in barang
            orderby x.Kode
            select new {
                Kode = x.Kode,
                Nama = x.Nama
            };
};
```

Baca setiap nilai (diwakili oleh "x") dari variabel "**barang**", dimana pada setiap nilai "x" akan terdiri dari 3 (tiga) variabel, yakni variabel "**Kode**", "**Nama**", dan "**Jenis**",  
Kemudian diurutkan berdasarkan variabel "**Kode**" dari nilai "x" secara Ascending,  
Kemudian hasil pengurutan tersebut hanya dikembalikan (return) variabel "**Kode**" dan "**Nama**" dari nilai "x" ke variabel "**hasil**".

- Untuk **Contoh Sederhana ke-3**, deklarasikan struktur berikut pada Console Application :

```
struct Faktur {
    public int Kode;
    public string Nama;
    public int Qty;
    public int HargaSatuan;
}
```

## PENJELASAN DARI SLIDE ke-21 & 22

Kemudian tuliskan kode program berikut :

```
List<Faktur> barang = new List<Faktur>() {
    new Faktur() {Kode = 67, Nama = "Gula", Qty = 5, HargaSatuan = 15000},
    new Faktur() {Kode = 34, Nama = "Beras", Qty = 2, HargaSatuan = 125000},
    new Faktur() {Kode = 98, Nama = "Minyak", Qty = 8, HargaSatuan = 17000},
    new Faktur() {Kode = 13, Nama = "Tepung", Qty = 12, HargaSatuan = 8000},
    new Faktur() {Kode = 51, Nama = "Daging", Qty = 4, HargaSatuan = 100000}
};
var hasil = from x in barang
let subtotalawal = x.Qty * x.HargaSatuan
let diskon = subtotalawal > 200000 ? subtotalawal * 0.1f : 0
let subtotalakhir = subtotalawal - diskon
select new {
    Nama = x.Nama,
    Diskon = diskon,
    SubTotal = subtotalakhir
};
foreach (var i in hasil)
    Console.WriteLine("{0,-8}{1,10} (Diskon {2})",
        i.Nama, i.SubTotal, i.Diskon);
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

(1)

Dari setiap nilai yang dibaca (diwakili oleh "x") dari variabel "barang",

Kemudian dikalikan variabel "Qty" dengan "HargaSatuan", dan disimpan ke variabel lokal "subtotalawal"

```
var hasil = from x in barang
let subtotalawal = x.Qty * x.HargaSatuan
let diskon = subtotalawal > 200000 ? subtotalawal * 0.1f : 0
let subtotalakhir = subtotalawal - diskon
select new {
    Nama = x.Nama,
    Diskon = diskon,
    SubTotal = subtotalakhir
};
```

(4)

Dikembalikan (return) ke variabel "hasil" berupa 3 (tiga) nilai, yakni :

1. Nilai dari variabel "Nama" dari nilai "x"
2. Besaran diskon dari variabel lokal "diskon" (2)
3. Hasil Sub Total akhir dari variabel lokal "subtotalakhir"

(2)

Dilakukan percabangan dengan operator ternary, jika nilai variabel lokal "subtotalawal" > 200000, maka dihitung diskon sebesar 10%, jika tidak, maka diskon = 0. Besaran diskon disimpan variabel lokal "diskon"

(3)

Dihitung Sub Total akhir setelah dipotong diskon, dan disimpan ke variabel lokal "subtotalakhir"

## PENJELASAN DARI SLIDE ke-23 & 24

- Operator **Any** dan **All** merupakan method LINQ untuk mengecek apakah sebagian atau semua data dari sumber data (Data Source) memenuhi suatu kriteria.
- Operator **Any** untuk mengecek apakah ada tidaknya (sebagian) data dari sumber data (Data Source) yang memenuhi kriteria. Jika ada 1 (satu) data saja yang memenuhi kriteria, maka akan mengembalikan nilai TRUE, sebaliknya jika semua data tidak ada yang memenuhi kriteria, maka akan mengembalikan nilai FALSE.
- Operator **All** untuk mengecek apakah semua data dari sumber data (Data Source) memenuhi kriteria atau tidak. Jika semua data memenuhi kriteria, maka akan mengembalikan nilai TRUE, sebaliknya jika ada 1 data saja yang tidak memenuhi kriteria, maka akan mengembalikan nilai FALSE.
- Operator **Any** dan **All** hanya dapat digunakan pada Method Syntax.

### [Contoh Sederhana]

- Contoh sederhana dari penggunaan operator **Any** dan **All** adalah menggunakan Console Application.
- Untuk **Contoh Sederhana ke-1**, deklarasikan struktur berikut pada Console Application :

```
List<int> data = new List<int>() { 21, 13, -5, 17, -33 };
Console.WriteLine(data.Any(i => i < 0));
Console.WriteLine(data.Any(i => i > 0));
Console.WriteLine(data.Any(i => i == 0));
Console.WriteLine(data.All(i => i % 2 != 0));
Console.WriteLine(data.All(i => i % 2 == 0));
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

1. Untuk potongan kode program LINQ berikut :
 

```
Console.WriteLine(data.Any(i => i < 0));
```

 Mengecek apakah ada tersimpan nilai negatif di dalam variabel **"data"** ?  
 Pada contoh kode program ini, terdapat nilai negatif, yakni {-5, -33}, maka akan mengembalikan nilai TRUE.
2. Untuk potongan kode program LINQ berikut :
 

```
Console.WriteLine(data.Any(i => i > 0));
```

 Mengecek apakah ada tersimpan nilai positif di dalam variabel **"data"** ?  
 Pada contoh kode program ini, terdapat nilai positif, yakni {21, 13, 17}, maka akan mengembalikan nilai TRUE.
3. Untuk potongan kode program LINQ berikut :
 

```
Console.WriteLine(data.Any(i => i == 0));
```

 Mengecek apakah ada tersimpan nilai NOL di dalam variabel **"data"** ?  
 Pada contoh kode program ini, tidak terdapat nilai NOL, maka akan mengembalikan nilai FALSE.
4. Untuk potongan kode program LINQ berikut :
 

```
Console.WriteLine(data.All(i => i % 2 != 0));
```

 Mengecek apakah semua nilai yang tersimpan di dalam variabel **"data"** adalah bilangan ganjil ?  
 Pada contoh kode program ini, semuanya adalah bilangan ganjil, maka akan mengembalikan nilai TRUE.
5. Untuk potongan kode program LINQ berikut :
 

```
Console.WriteLine(data.All(i => i % 2 == 0));
```

 Mengecek apakah semua nilai yang tersimpan di dalam variabel **"data"** adalah bilangan genap ?  
 Pada contoh kode program ini, tidak ada bilangan genap sama sekali, maka akan mengembalikan nilai FALSE.

## PENJELASAN DARI SLIDE ke-23 &amp; 24

- Untuk **Contoh Sederhana ke-2**, deklarasikan struktur berikut pada Console Application :

```
List<string> nama = new List<string>()
{ "sally", "alex", "meri", "yohan", "donita" };
Console.WriteLine(nama.Any(i => i.Contains('a')));
Console.WriteLine(nama.All(i => i.Contains('a')));
Console.WriteLine(nama.Any(i => i.StartsWith("d")));
Console.WriteLine(nama.All(i => i.EndsWith("i")));
```

Potongan kode program LINQ di atas memiliki penjelasan berikut :

1. Untuk potongan kode program LINQ berikut :

```
Console.WriteLine(nama.Any(i => i.Contains('a')));
```

Mengecek apakah ada tersimpan nilai yang mengandung karakter **"a"** di dalam variabel **"nama"**?  
Pada contoh kode program ini, terdapat nilai yang mengandung karakter **"a"**, yakni {"sally", "alex", "yohan", "donita"}, maka akan mengembalikan nilai TRUE.

2. Untuk potongan kode program LINQ berikut :

```
Console.WriteLine(nama.All(i => i.Contains('a')));
```

Mengecek apakah semua nilai yang tersimpan di dalam variabel **"nama"** mengandung karakter **"a"**?  
Pada contoh kode program ini, terdapat 1 (satu) nilai yang tidak mengandung karakter **"a"** yakni {"meri"}, maka akan mengembalikan nilai FALSE.

3. Untuk potongan kode program LINQ berikut :

```
Console.WriteLine(nama.Any(i => i.StartsWith("d")));
```

Mengecek apakah ada tersimpan nilai yang diawali dengan karakter **"d"** di dalam variabel **"nama"**?  
Pada contoh kode program ini, terdapat nilai yang diawali dengan karakter **"d"**, yakni {"donita"}, maka akan mengembalikan nilai TRUE.

4. Untuk potongan kode program LINQ berikut :

```
Console.WriteLine(nama.All(i => i.EndsWith("i")));
```

Mengecek apakah semua nilai yang tersimpan di dalam variabel **"nama"** diakhiri dengan karakter **"i"**?

Pada contoh kode program ini, hanya terdapat 1 (satu) yang diakhiri dengan karakter **"i"** yakni {"meri"} dari total 5 (lima) nilai yang tersimpan di dalam variabel **"nama"**, maka akan mengembalikan nilai FALSE.